# A Hash, a Clock and a Signature

*The Lightning Network built from nothing — a channel worked through five states, the hash time-locked contract, a three-hop route with every fee and timelock, liquidity as probability — then the bridge to M-Pesa, and the Malipo Rail re-specified.*

**Dr Neal Aggarwal**

Medicine · Surgery · Engineering · Information Technology · Artificial Intelligence

2 NOVEMBER 2026

In September I published a working note about Malipo Rail, the payment system I am designing for the Kenya Institute for Clinical AI, and called it a draft 'while it is still wet'. It asserted that Lightning would carry the Institute's money and never explained how Lightning works. This is the full piece. It builds the Lightning Network from nothing, with numbers you can check: a payment channel between a clinic and the Institute, worked through five states, including what happens when one side cheats; the hash time-locked contract, with a real SHA-256 hash, that lets a payment hop across strangers without trusting any of them; a three-hop route with every fee and every timelock computed backwards from the destination; onion routing; why a payment can fail when every channel on the route is big enough, and the probability model that explains it; and liquidity — the operational problem that decides whether any of this works. It then crosses into shillings: how M-Pesa actually moves money, what the Daraja API exposes, and how a hold invoice can be wired to an M-Pesa callback so that a bitcoin payment and a shilling payout either both happen or neither does. Finally it re-specifies Malipo Rail in the light of all of it — including the places where working it through by hand showed that my own draft was wrong — and brings the Kenyan regulatory picture up to date, two days before the licensing deadline.

# Contents


Download this essay as a PDF 5

Part 1 — What problem Lightning actually solves 8

Part 2 — A payment channel, by hand 9

- Step 1 — Opening the channel 9
- Step 2 — Paying by replacing the commitment 9
- Step 3 — The fatal flaw 10
- Step 4 — The fix: make old states into traps 10
- Step 5 — What happens when Clinic cheats 11
- Step 6 — The details that matter in practice 12

Part 3 — The hash time-locked contract, by hand 14

- The hash, worked 14
- The contract 14
- Why the same hash makes a route safe 15

Part 4 — A three-hop route, by hand 17

- The amounts 17
- The timelocks 17
- The sequence, message by message 18

Part 5 — Onion routing: each node sees its neighbours and nothing else 20

Part 6 — Why payments fail: liquidity, by hand 21

- The probability model 21
- Inbound liquidity: the problem nobody expects 22

Part 7 — What can go wrong 24

Part 8 — From satoshis to shillings: M-Pesa, by the same method 25

- What M-Pesa is, mechanically 25
- The Daraja API 25
- The bridge problem 26
- The hold invoice: making the bridge almost atomic 26
- The incumbents already doing this 27
- The comparison I promised in September 27

Part 9 — The Malipo Rail, re-specified 29

Three corrections to my own draft 29

The node, the LSP and the treasury 30

What the attestation registry records 30

What does not change 31

**Part 10 — The Kenyan regulatory picture, two days before the deadline** 32

**Part 11 — The decisions log, updated** 33

**What this does not do** 34

**Where this leaves the argument** 34

**A short glossary** 35

**Further reading and references** 37

# A Hash, a Clock and a Signature

## The Lightning Network, by hand — and the Malipo Rail

How a payment crosses strangers without trusting any of them — and how it lands as shillings on an M-Pesa phone.

THE CHANNEL · PENALTY

### Old states become traps

Signed, unbroadcast transactions split a 2-of-2 output. Each update hands over the secret for the last state. Broadcast an old one and you lose everything.

THE HTLC · HASH + CLOCK

### Paid if you know the secret, in time

Same hash at every hop; deadlines shrink each step. Reveal the preimage and every hop can be paid. Never reveal it and every hop gets its money back.

THE ROUTE · LIQUIDITY

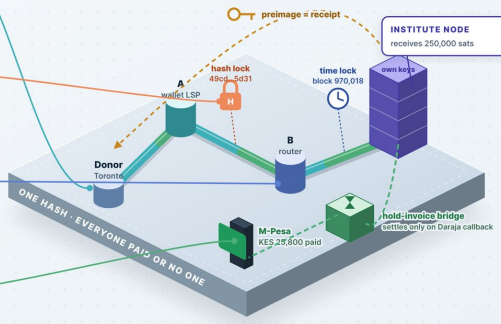
### Capacity is public; balance is not

Fees and timelocks computed back from the payee. 176 sats to move 250,000 over three hops. Success is a probability: P = (c + 1 − a)/(c + 1).

THE BRIDGE · M-PESA

### Hold the payment, then pay the phone

Bridge holds the HTLC, calls Daraja B2C. Success callback: reveal preimage, take the sats. Failure: cancel, and the sats go home.




**A PROMISE THAT SETTLES ITSELF.**

One hash at every hop: either everyone on the route is paid, or no one is.

### THE MALIPO RAIL, RE-SPECIFIED

1. **Patients pay in M-Pesa**
   Cheaper than any route through bitcoin
2. **Donations from abroad: Lightning**
   ~0.07% in routing fees, final in a second
3. **Contractors: hold-invoice bridge**
   Sats move only once the shillings land
4. **Reserves: cold, on-chain**
   Out of channels, out of the quantum line

### WHAT KENYAN LAW HASN'T SAID

as of the end of October 2026

ASK COUNSEL

D-01
o Is a self-custodial node a VASP? Unresolved

DEADLINE
! Transitional period ends 4 November 2026

LICENCES
0 None listed by regulators in late October

STABLECOINS
× CBK may restrict foreign dollar stablecoins

drnealaggarwal.info · 2026

> *I write here in a personal capacity. This is the full write-up promised in What I'm Building Next: A Bitcoin Rail for the Clinical AI Institute, and it belongs to the same series as Not Shown Is Not Locked, Twenty Generals, One Ledger, The Spinning Coin and the Stolen Key and Learning With Errors, by Hand. Every number in the worked examples was computed by hand and checked in code. Bitcoin prices are illustrative and stated where used. Nothing here is legal, regulatory, tax or investment advice, and Malipo Rail remains a design: nothing has been built and no money has moved.*

---

## Download this essay as a PDF

The whole argument as a printable essay — a payment channel worked through five states, the hash time-locked contract with a real hash, a three-hop route with every fee and timelock computed, the liquidity probability model, the hold-invoice bridge to M-Pesa, and the re-specified Malipo Rail with its updated decisions log.

**Aggarwal N. *A Hash, a Clock and a Signature: The Lightning Network, by Hand, and the Malipo Rail.* 2026.**

**Download PDF (602 KB, 37 pages, 3 figures)**

**Listen to this post (62 minutes):** The full conversation around this essay, from the first spark to the finished piece. Hosted privately on Google Drive.

Listen online at drnealaggarwal.info

---

A donor in Toronto wants to give the Institute forty dollars a month. A contractor in Mombasa has finished a week’s work on the Institute’s assessment software and wants to be paid in shillings, on her phone, today. A clinic in Kisumu owes its quarterly dues. Three payments, three different shapes of problem — and in September I published a note saying that the Lightning Network would carry all of them, without once explaining what the Lightning Network is or why anyone should trust it to.

That note called itself “a working note, not a finished piece … while it is still wet”, and promised a proper write-up. This is it. I want to do for Lightning what I did for lattice cryptography in Learning With Errors, by Hand and for consensus in Twenty Generals,

One Ledger: build it from nothing, with numbers small enough to check, until the claims stop being assertions.

The claims are bigger than they look. Lightning says that two people can pay each other thousands of times a second without telling anyone, that a payment can travel through a chain of strangers none of whom can steal it, and that the whole thing settles in a few hundred milliseconds for a fraction of a cent. Each of those sounds impossible in exactly the way a perpetual-motion machine sounds impossible. Each is achieved by combining three primitive objects that Bitcoin has had since 2009 or soon after: **a hash, a clock and a signature**. By the end of this piece you will have built all three into a channel, a contract and a route by hand, and then wired the result to M-Pesa.

Working it through also changed my design. Three things in my September draft turn out to be wrong or overstated once you do the arithmetic, and I say which as I reach them.

**What this post explains, in one page**

**The channel.** Two parties lock bitcoin into an output that needs both their signatures to spend. They then exchange signed-but-unbroadcast transactions that split it between them, replacing each with the next as they pay each other. Only the first and last transactions ever touch the blockchain.

**The penalty.** Every old state remains a valid transaction forever, so each side hands the other a secret that makes broadcasting an old state suicidal: the cheat's own funds are delayed by a timelock, and the victim can use the secret to take all of them in the meantime.

**The HTLC.** A hash time-locked contract says: "this money is yours if you show me the number that hashes to $H$ before block $T$; after that it comes back to me." Chain several of them with the same $H$ and decreasing $T$, and a payment can cross any number of strangers atomically: either every hop is paid or none is.

**The route.** Fees and timelocks are computed backwards from the destination. Each node is paid for forwarding and given a safety margin in blocks to claim its incoming money after paying out the outgoing. Onion encryption means each node sees only its two neighbours.

**Liquidity.** The network publishes how big each channel is, never how the balance inside it is split. Whether a payment succeeds is therefore a probability, and the single most important design decision for an organisation joining Lightning is how it gets money onto the far side of its own channels.

**The bridge.** M-Pesa is not a blockchain and its payments are not conditional. But a *hold invoice* lets a bridge accept a Lightning payment provisionally, trigger an M-Pesa payout through Safaricom's Daraja API, and only release the hash secret — completing the Lightning payment — once Safaricom confirms the shillings have landed.

> **Malipo Rail, re-specified.** Patients should pay in M-Pesa and never touch Lightning at all. Lightning earns its place for donor money arriving from abroad, for paying contractors, and for settlement between institutions. The treasury stays in cold multisignature storage, never in channels. And the Kenyan licensing question I flagged in September is still open, two days before the deadline.

---

## Part 1 — What problem Lightning actually solves

Start with what Bitcoin’s base layer is good and bad at.

It is good at final settlement between parties who do not trust each other. A transaction buried under six blocks is, for every practical purpose, irreversible, and nobody — no bank, no government, no Institute board — can reverse or censor it. That is the property the Institute’s treasury needs, and the reason I chose Bitcoin in the first place.

It is bad at everything a clinic payment needs. A block arrives on average every ten minutes, so a payment waits minutes for its first confirmation and about an hour for six. Each block holds a few thousand transactions, so the whole system settles somewhere in the low single digits of transactions per second, worldwide. And because block space is scarce, it is auctioned: when demand spikes, the fee for one transaction can exceed the value of a small payment. A system that might charge three dollars to move four is not a payment system for clinics.

The obvious fix is IOUs. Two parties who transact often can keep a running tab off the chain and settle the net balance occasionally. Banks have done this for centuries. The problem is trust: an IOU is only as good as the person who wrote it, and the whole point of using Bitcoin was not to need that.

Lightning’s insight is that the IOUs can be **real Bitcoin transactions** — fully signed, valid, broadcastable at any moment — that the parties simply choose not to broadcast. Each one is a better IOU than any bank’s: it does not depend on the other party being solvent, honest or even alive, because either side can take it to the blockchain unilaterally and the blockchain will honour it. The parties pay each other by exchanging new versions. Only the opening and the final settlement are ever recorded on-chain.

That idea has one fatal flaw, which Part 2 builds up to and then fixes. Everything else in Lightning is a consequence of the fix.

## Part 2 — A payment channel, by hand

The cast: **Clinic**, a member clinic in Kisumu, and **Hub**, the Institute's Lightning node in Nairobi. I will use satoshis throughout — one bitcoin is 100,000,000 satoshis ("sats") — and, where I need a shilling figure, an illustrative price of US$80,000 per bitcoin (roughly where it traded in late October) and KES 129 to the dollar. At those rates one satoshi is about 0.1 shillings.

### Step 1 — Opening the channel

Clinic puts **1,000,000 sats** (about KES 103,000) into a new on-chain transaction: the **funding transaction**. Its single important output can only be spent by a transaction signed by *both* Clinic's key and Hub's key — a 2-of-2 multisignature output.

On its own, that would be a terrible idea for Clinic. If Hub disappeared, the money would be locked forever. So *before* Clinic broadcasts the funding transaction, Hub signs a second transaction that spends the funding output and gives all 1,000,000 sats back to Clinic. Clinic holds that signed transaction privately. Only then does Clinic broadcast the funding transaction.

This ordering is the whole trick in miniature. Clinic never puts money anywhere it cannot get it back from, because it holds a signed, valid exit before it enters. That exit is called a **commitment transaction**, and the channel is nothing more than a succession of them.

### Step 2 — Paying by replacing the commitment

Clinic pays its dues of 300,000 sats to the Institute. It does not touch the blockchain. Instead the two sides create and sign a new commitment transaction spending the same funding output:

| State | Clinic | Hub | What happened |
|---|---|---|---|
| 0 | 1,000,000 | 0 | Channel opened |
| 1 | 700,000 | 300,000 | Clinic pays quarterly dues |
| 2 | 650,000 | 350,000 | Clinic pays a CPD fee |
| 3 | 750,000 | 250,000 | Institute refunds an overpayment |
| 4 | 400,000 | 600,000 | Clinic pays for a cohort's certification |

Each row is a signed transaction that either side could broadcast at any moment. The latest one is the agreed balance. When they are finished — next year, or never — they sign a final **cooperative close** that pays out the latest balance, broadcast it, and the blockchain sees exactly two transactions: the funding transaction and the close. Four payments, or four million, in between: the chain neither knows nor cares.

### Step 3 — The fatal flaw

Look at state 0 again. It is a fully signed transaction paying Clinic 1,000,000 sats. Clinic still has it. Nothing about signing state 4 makes state 0 invalid — Bitcoin has no concept of "this transaction has been superseded by an agreement you made privately". So at state 4, when Clinic's real balance is 400,000, Clinic can broadcast state 0 and walk away with 1,000,000.

Every off-chain payment scheme before Lightning ran aground here. There is no way to *delete* a signed transaction. Lightning does not try. It makes broadcasting an old one so dangerous that nobody rational would.

### Step 4 — The fix: make old states into traps

Two changes, both using the "clock" and "signature" from the title.

**First, the two sides hold different versions of each state.** Clinic's copy of state 4 and Hub's copy of state 4 pay the same amounts, but they are built so that whoever *broadcasts* a commitment gets their own money only after a delay. In Clinic's copy of state 4, Hub's 600,000 is spendable by Hub immediately, but Clinic's own 400,000 sits in an output that says:

> Clinic can spend this, but only after the transaction has been buried under **144 blocks** (about a day) — *or* anyone holding the **revocation key for this state** can spend it immediately.

Hub's copy is the mirror image. The 144-block delay is called `to_self_delay` and is negotiated when the channel opens; values from about a day to two weeks are common, longer for larger channels. It is implemented with Bitcoin's relative timelock, `OP_CHECKSEQUENCEVERIFY`: the clock in the title.

For readers who want the real thing, this is the script of that delayed output as the specification (BOLT 3) gives it:

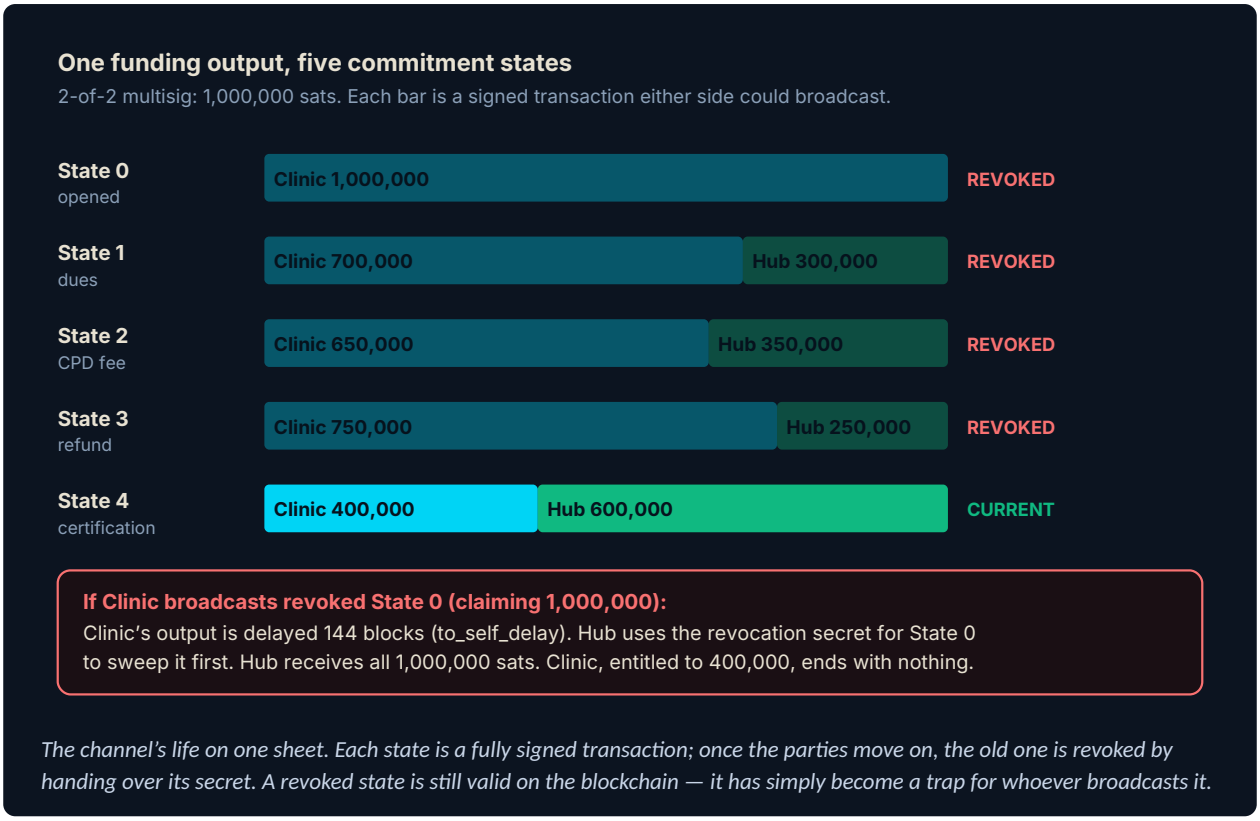
```
OP_IF
    # Penalty transaction
    <revocationpubkey>
OP_ELSE
    `to_self_delay`
    OP_CHECKSEQUENCEVERIFY
    OP_DROP
    <local_delayedpubkey>
OP_ENDIF
OP_CHECKSIG
```

Read it as two doors. The first opens immediately for whoever can sign with `revocation-pubkey`. The second opens for the broadcaster's own key, but only after `to_self_delay` blocks have passed since the commitment confirmed. The other party's balance in the same transaction is paid to a plain key with no delay at all.

"The secret for the old state" is, in the real protocol, a **per-commitment secret**. Each commitment uses fresh keys derived from a long-lived *basepoint* and a per-commitment

*point*; `revocationpubkey` combines the counterparty's revocation basepoint with the broadcaster's per-commitment point in such a way that neither party alone can compute the matching private key. When the broadcaster later reveals the per-commitment secret, the counterparty can finally compute it. That is why revealing one 32-byte number is enough to hand over the power to punish — and why it can only ever punish *that* state.

**Second, when the parties move to a new state, each hands the other the secret for the old one.** When Clinic and Hub sign state 5, Clinic gives Hub the secret from which the revocation key for Clinic's state 4 can be derived, and Hub gives Clinic the equivalent for Hub's. Once handed over, the old state is *revoked*: still a valid transaction, but one whose broadcaster has given the other party the means to take everything.




*The channel's life on one sheet. Each state is a fully signed transaction; once the parties move on, the old one is revoked by handing over its secret. A revoked state is still valid on the blockchain — it has simply become a trap for whoever broadcasts it.*

### Step 5 — What happens when Clinic cheats

Suppose that at state 4 someone at Clinic is tempted. They broadcast Clinic's copy of **state 0**, which pays Clinic 1,000,000 sats.

- The transaction confirms. Hub's output in state 0 was zero, so Hub gets nothing from it directly.
- Clinic's 1,000,000 sits in the delayed output. Clinic cannot touch it for 144 blocks.
- Hub's node, watching the blockchain, recognises a revoked state. It holds the revocation secret for Clinic's state 0, which Clinic handed over when they moved to state 1.
- Hub derives the revocation key and, well within the 144 blocks, spends Clinic's delayed output to itself.

Result: Hub receives **all 1,000,000 sats**. Clinic, which was entitled to 400,000, ends with nothing. The attempted theft of 600,000 costs the thief 400,000 of their own money.

That is the whole security model of a channel. It does not stop cheating; it makes cheating a bet whose downside is everything you had in the channel and whose upside depends on the victim being asleep for a day. The incentive runs entirely one way.

> **The catch: someone has to be watching**
>
> The penalty only works if the victim notices within `to_self_delay`. A node that is offline for longer than the delay can be robbed. For an institution this is an operational requirement, not a theoretical one: the Institute's nodes must be monitored continuously, and should also register with independent **watchtowers** — third-party services that are handed encrypted penalty transactions and broadcast them if they see a revoked state confirm. A watchtower learns nothing about the channel until a breach happens. The cost of this is real staff time and infrastructure; it is one of the reasons Part 9 does not put the Institute's reserves in channels.

### Step 6 — The details that matter in practice

A few things the clean table above hides, each of which a finance officer will eventually ask about.

**Storing every old secret sounds impossible.** It would be, for a channel with millions of states. In practice the secrets are generated from a single seed by a scheme (called "shachain" in the specification) that lets the receiver store only 49 values to reconstruct any of $2^{48}$ past secrets. Storage is constant, not proportional to the number of payments.

**Each side keeps a reserve.** The protocol requires each side to keep a small balance — commonly 1% of the channel, so 10,000 sats here — that it cannot spend. Otherwise a party whose balance had fallen to zero would have nothing to lose by broadcasting an old state.

**Fees for a transaction you might broadcast in a year.** A commitment transaction has to pay an on-chain fee when broadcast, but nobody knows today what fee will be needed in a year. Modern channels solve this with **anchor outputs**: each commitment carries two tiny outputs (330 sats each) that either side can spend to attach a higher fee at the moment of broadcast ("child pays for parent"). This is why a force-closed channel returns slightly less than its balance.

**Two ways to close.** A *cooperative* close is one cheap transaction paying both sides immediately. A *force* close — one side broadcasting its latest commitment unilaterally, because the other has vanished or misbehaved — delays the broadcaster's own funds by `to_self_delay` and costs more in fees. Force closes are the commonest way operators lose money through mismanagement rather than theft.

**Resizing without closing.** Until recently, adding money to a channel or withdrawing some meant closing it and opening a new one. **Splicing** — replacing the funding output with a new one of a different size while the channel stays open — is now on by default in Core Lightning and partly supported elsewhere, though not yet in LND. For an institution whose flows grow over time, it matters.

---

## Part 3 — The hash time-locked contract, by hand

A channel lets two parties pay each other. To pay someone you have no channel with, a payment has to cross other people's channels. The donor in Toronto has a channel with her wallet provider; the provider has a channel with a routing node; the routing node has one with the Institute. Why should any of them forward money for her? What stops the routing node taking the money and not passing it on?

The answer is the third primitive: **the hash**.

### The hash, worked

A cryptographic hash function takes any input and produces a fixed-size fingerprint. Lightning uses SHA-256, which produces 256 bits — 64 hexadecimal characters. Its two relevant properties are that it is easy to compute in one direction and, as far as anyone knows, impossible to reverse: given the output, nobody can find an input that produces it except by guessing.

When the Institute wants to be paid, its node picks a secret, the **preimage**, and computes its hash. In practice the preimage is 32 random bytes. To keep it printable, here is the same operation on a short phrase:

> SHA-256("Malipo invoice 0001") =
> `49cdfe288e47a57ce2a5c94424811fbda610a830f038b44b325a0cf8682a5d31`
>
> SHA-256("Malipo invoice 0002") =
> `fd017b7e373fce29d3d6356a2016d7d288808b9493b48bb867945c77258b13d7`

You can check both on any computer. Change one character and the output bears no resemblance to the first. Given only the first line of hex, the only way to find a phrase that produces it is to try phrases until one fits — which, for a random 32-byte preimage, means more guesses than there are atoms in the observable universe.

The Institute keeps the preimage secret and puts the hash in an **invoice**: "Pay 250,000 sats to the payment hash `49cd…5d31`." That invoice is the entire instruction the donor's wallet needs.

### The contract

A **hash time-locked contract** (HTLC) is a conditional payment added to a channel as a third output of the commitment transaction, alongside the two balances. In plain language, the output that Toronto's wallet provider offers onward says:

> These 250,000 sats go to the next node **if it can produce the preimage of `49cd…5d31`**, together with its signature. **If it has not done so by block *T*,** they come back to me.

The specification's version of this output — an *offered* HTLC, in its original form without anchor outputs — is:

```
# To remote node with revocation key
OP_DUP OP_HASH160 <RIPEMD160(SHA256(revocationpubkey))> OP_EQUAL
OP_IF
    OP_CHECKSIG
OP_ELSE
    <remote_htlcpubkey> OP_SWAP OP_SIZE 32 OP_EQUAL
    OP_NOTIF
        # To local node via HTLC-timeout transaction (timelocked).
        OP_DROP 2 OP_SWAP <local_htlcpubkey> 2 OP_CHECKMULTISIG
    OP_ELSE
        # To remote node with preimage.
        OP_HASH160 <RIPEMD160(payment_hash)> OP_EQUALVERIFY
        OP_CHECKSIG
    OP_ENDIF
OP_ENDIF
```

Three paths, read top to bottom. If this commitment has been revoked, the counterparty takes the HTLC with the revocation key — the penalty again. Otherwise, if the spender supplies a 32-byte value (`OP_SIZE 32 OP_EQUAL`) whose hash matches, the downstream node takes the money with its own signature: that is the hash lock. If not, the money can return to the offerer, but only through a separate pre-signed **HTLC-timeout** transaction that both parties signed in advance and that is itself locked until block $T$: that is the time lock. The second-stage transaction exists so that the broadcaster's refunded money still passes through the `to_self_delay` penalty window, and cannot be used to sidestep it. Note the small economy in the script: it stores `RIPEMD160` of the payment hash, 20 bytes rather than 32, and checks the preimage against it with `OP_HASH160`, which is SHA-256 followed by RIPEMD-160.

Three primitives in one sentence. The *hash* makes the payment conditional on knowledge of a secret. The *clock* — this time an absolute timelock, `OP_CHECKLOCKTIMEVERIFY`, naming a block height — guarantees the money comes back if the secret never appears. The *signatures* make sure only the intended recipient can claim it.

In a healthy network this output is never broadcast. The two sides simply agree that the condition has been met (the preimage has been revealed) or has failed, and fold the HTLC's amount into the ordinary balances in the next commitment. The script exists so that either side can enforce the agreement on-chain if the other stops cooperating.

### Why the same hash makes a route safe

Here is the property that makes routing possible. Suppose every channel along the route carries an HTLC locked to the **same hash**. The Institute, at the end, is the only party who knows the preimage. To claim its incoming HTLC, it must reveal the preimage to the node before it. That node now knows the preimage, so it can claim *its* incoming HTLC from the node before it — and so on, back to the donor.

- If the Institute reveals the preimage, every node on the path can be paid, and each is paid exactly when it has been paid out.

- If the Institute never reveals it, every HTLC eventually times out and every node gets its money back.

There is no state in which a forwarding node has paid out without being able to claim the incoming payment, provided it acts in time. The routing node cannot steal the payment, because the money it holds is locked to a secret it does not know. And the preimage doubles as a receipt: the donor’s wallet ends up holding a number that only the Institute could have revealed, and only by accepting the payment. That is proof of payment, and Part 9 uses it.

---

## Part 4 — A three-hop route, by hand

Now a real route, with every number computed. The donor (**Donor**) pays the Institute's node (**Institute**) 250,000 sats — about US$200, or KES 25,800 at the illustrative rates. Donor has a channel with her wallet provider's node **A**; A has a channel with routing node **B**; B has one with Institute.




Each forwarding node publishes two things about each of its channels: a **fee policy** (a flat base fee plus a proportional fee in parts per million) and a **timelock delta**, the number of blocks of safety margin it demands between the HTLC it receives and the one it sends on. The values below are illustrative but typical; implementations' default deltas range from 34 blocks (Core Lightning) and 36 (LDK) through 80 (LND) to 144 (Eclair).

| Node | Fee on its outgoing channel | Timelock delta |
|---|---|---|
| A (wallet provider, Core Lightning) | 1 sat base + 500 ppm | 34 blocks |
| B (routing node, LND) | 0 base + 200 ppm | 80 blocks |

Fees are computed in millisatoshis (thousandths of a satoshi), which is what the protocol uses internally. The rule is to **work backwards from the destination**, because each node's fee is charged on the amount it sends onward.

### The amounts

1. **Institute must receive** 250,000 sats = 250,000,000 msat.
2. **B forwards** that to Institute and charges 0 + 200 ppm of 250,000,000 = **50,000 msat** (50 sats). So **A must send B** 250,050,000 msat.
3. **A forwards** 250,050,000 to B and charges 1,000 msat + 500 ppm of 250,050,000 = 1,000 + 125,025 = **126,025 msat**. So **Donor must send A** 250,176,025 msat.

Total fees: **176,025 msat — about 176 sats, 0.07% of the payment, around 14 US cents or KES 18.** For an international transfer of US$200 that arrives in under a second, that is the comparison that matters. Part 8 makes it against M-Pesa.

### The timelocks

Say the current block height is **970,000**. The Institute's invoice asks for a final timelock margin of 18 blocks, the default in the invoice standard. Working backwards again:

| HTLC | Amount (msat) | Expires at block | Margin over the next hop |
|---|---|---|---|
| B → Institute | 250,000,000 | 970,018 | — (final: 18 blocks from now) |
| A → B | 250,050,000 | 970,098 | 80 blocks (B's delta) |
| Donor → A | 250,176,025 | 970,132 | 34 blocks (A's delta) |

Read the right-hand column carefully, because it is where routing safety lives. B is sending an HTLC that expires at block 970,018 and receiving one that expires at 970,098. If Institute claims B's outgoing HTLC at the very last moment — even on-chain, at block 970,017 — B learns the preimage then and still has **80 blocks, about thirteen hours,** to claim its incoming HTLC from A, by force-closing on-chain if A will not cooperate. The delta is B's insurance against being paid out downstream and failing to collect upstream.

That is also why the deltas are not smaller. In a period of on-chain congestion, B's claim might take many blocks to confirm. A delta that is too short is an invitation to the attack in Part 7.




← preimage of 49cd…5d31 travels back; each hop claims its incoming HTLC

**What each forwarding node earns, and how long it has to collect**

| | | |
|---|---|---|
| A | 126,025 msat (≈126 sats) | receives at 970,132, pays at 970,098 → 34 blocks to claim upstream |
| B | 50,000 msat (50 sats) | receives at 970,098, pays at 970,018 → 80 blocks to claim upstream |
| Total | 176,025 msat ≈ 0.07% | ≈ US$0.14 on a US$200 payment (illustrative prices) |

Current block height assumed 970,000; final margin 18 blocks (the invoice default). Same hash at every hop.

*The three-hop route with every amount and expiry computed backwards from the Institute. HTLCs go out left to right, all locked to the same hash; the preimage comes back right to left, and each node claims its incoming payment with a margin of blocks to spare.*

## The sequence, message by message

The channel messages that make this happen are specified in the Lightning standards (the "BOLTs"). Stripped to the essentials:

1. **Donor → A:** `update_add_htlc` — 250,176,025 msat, hash `49cd…`, expiry 970,132, plus an encrypted onion (Part 5). Donor and A sign a new commitment including the HTLC.
2. **A → B:** `update_add_htlc` — 250,050,000 msat, same hash, expiry 970,098. New commitment.
3. **B → Institute:** `update_add_htlc` — 250,000,000 msat, same hash, expiry 970,018. New commitment.
4. **Institute** checks that the amount and expiry match its invoice and that it knows the preimage. It replies `update_fulfill_htlc` with the preimage. The HTLC is folded into the balances: Institute's side of the B–Institute channel rises by 250,000 sats.
5. **B** now knows the preimage. It sends `update_fulfill_htlc` to A. B's side of the A–B channel rises by 250,050 sats: B is up 50 sats net.

6. **A** sends `update_fulfill_htlc` to Donor. A is up 126 sats net. Donor's wallet now holds the preimage — the receipt.

On a healthy network this takes a few hundred milliseconds. Nothing touched the blockchain.

If anything goes wrong — Institute does not recognise the hash, B's channel to Institute lacks the funds, a node is offline — the failing node sends `update_fail_htlc` back along the path, each HTLC is removed in a new commitment, and the balances return to where they were. Donor's wallet then tries another route.

> **The stuck payment**
>
> There is a third outcome, and every Lightning user eventually meets it. If a node downstream simply goes silent after accepting the HTLC, neither fulfilling nor failing it, the payment is *stuck* until the timelock expires. The money is not lost — the clock guarantees it comes back — but it can be locked for hours, here up to 132 blocks, roughly a day. Wallets mitigate this by choosing reliable routes and splitting payments. For an institution, it is one more reason to have direct channels with dependable counterparties rather than relying on whatever the network offers.

---

## Part 5 — Onion routing: each node sees its neighbours and nothing else

In the sequence above, how did A know to forward to B, and B to Institute, without the whole route being visible to every node on it?

Donor's wallet constructs the route itself and wraps the instructions for each hop in successive layers of encryption, like an onion. The packet A receives is encrypted so that A can decrypt only its own layer: "forward 250,050,000 msat to the channel with B, with expiry 970,098, and pass on this inner packet." A cannot read the inner packet. B peels the next layer, and so on. The construction, called Sphinx, is a fixed size — a 1,300-byte payload — whatever the length of the route, so no node can infer its position from the packet's size, and each node's layer includes a check that detects tampering.

The result is that **A knows Donor sent it something and B is the next hop. B knows A and Institute, and does not know whether A was the origin.** Only Donor knows the whole path.

Two newer pieces of the standard extend this to the receiving side. **Blinded paths** let the recipient encrypt the last few hops of its own route, so that a payer can reach it without learning where its node sits in the network. **BOLT12 offers**, merged into the specification in September 2024, use blinded paths to provide a reusable, static payment code — the equivalent of a till number that can be printed once and paid many times, each time generating a fresh invoice with a fresh hash. BOLT12 is native in Core Lightning, LDK and Eclair; LND supports it only through a separate add-on.

> **Where the privacy leaks**
>
> The payment hash is the same at every hop. Two nodes on the same route that compare notes, or one entity running two nodes, can tell that the HTLCs they saw belong to one payment, and so can narrow down who paid whom. The fix — replacing hash locks with **point time-locked contracts** (PTLCs), which give each hop a different-looking lock using elliptic-curve arithmetic — has been designed for years and is not yet deployed. Lightning's privacy is much better than on-chain Bitcoin's, and much worse than its marketing. For Malipo Rail this matters less than it might, because the privacy that matters to patients is protected by never putting clinical context anywhere near a payment (Part 9).

## Part 6 — Why payments fail: liquidity, by hand

Every Lightning wallet has, at some point, failed to send a payment that it had the funds for, along a route of channels each of which was larger than the payment. To understand why, look at what the network actually publishes.

Each public channel is announced in a gossip network with its **capacity** — the size of the funding output, visible on the blockchain anyway. What is never published is the **balance**: how much of that capacity currently sits on each side. A channel of 1,000,000 sats might be 999,000 on one side and 1,000 on the other. It can only forward 250,000 sats in a given direction if at least that much sits on the sending side.

### The probability model

Richard Pickhardt and Stefan Richter formalised this in 2021. If you know nothing about where the balance sits, the simplest honest assumption is that every split from 0 to the capacity $c$ is equally likely. The probability that a channel can forward an amount $a$ is then:

> $$P(a) = (c + 1 - a) / (c + 1)$$

For a whole route, assuming the channels' balances are independent, the probabilities multiply.

Apply it to our route. Donor's own channel to A she can see — she knows her own balance — so it does not count. The other two:

| Channel | Capacity $c$ | Amount $a$ | P(a) |
|---|---|---|---|
| A → B | 2,000,000 | 250,000 | 0.875 |
| B → Institute | 1,000,000 | 250,000 | 0.750 |
| **Route** | | | **0.656** |

A route on which every channel is at least four times the payment succeeds, under this model, about **two times in three**. The weakest channel dominates: the last hop into the Institute alone is a 25% chance of failure.

Now change the one thing the Institute controls. If the channel into the Institute were 5,000,000 sats instead of 1,000,000:

| Channel | Capacity | Amount | P(a) |
|---|---|---|---|
| A → B | 2,000,000 | 250,000 | 0.875 |
| B → Institute | 5,000,000 | 250,000 | 0.950 |
| **Route** | | | **0.831** |

The same payment along the same path now succeeds five times in six. Halving the payment to 125,000 sats on the original channels gives 0.9375 × 0.875 = **0.820**, about the same improvement — which is why wallets split large payments into several parts sent along different routes (**multi-path payments**), all locked to the same hash so that the recipient claims either all of them or none.

Two further refinements make real routing much better than these figures. First, **failures are information**: if a 250,000-sat attempt fails at B → Institute, the sender now knows that channel's sending balance is below 250,000 and stops trying it for that amount. Second, modern pathfinding treats routing as a **min-cost flow** problem that trades off fees against the probability of success across many possible paths at once, rather than picking the cheapest single route and hoping. Core Lightning, LND and LDK all now use probability-weighted pathfinding of this kind.

### Inbound liquidity: the problem nobody expects

Here is the consequence that matters most for an institution, and it surprised me when I first ran a node.

When the Institute opens a channel with its own money — say 5,000,000 sats to a well-connected node — all 5,000,000 sit on **the Institute's side**. That channel can send 5,000,000 sats. It can **receive nothing**, because there is nothing on the far side to move towards the Institute. A new node that opens channels with its own funds is a node that can pay but cannot be paid.

For an organisation whose main use of Lightning is *receiving* — donations, dues, fees — this is the central operational problem, and it is solved in one of four ways:

1. **Spend first.** Pay out through the channel (contractors, suppliers), which moves balance to the far side and creates room to receive. Self-correcting for an institution whose outflows roughly match its inflows.
2. **Ask someone to open a channel to you.** A **Lightning Service Provider** (LSP) opens a channel *towards* the Institute with the LSP's own funds, for a fee. Standardised request formats (the LSPS specifications) and **just-in-time channels**, opened automatically at the moment a payment arrives, have made this routine.
3. **Buy it in a market.** Liquidity advertisements and leasing let a node pay for inbound capacity for a fixed period.
4. **Swap it.** A submarine swap or loop-out sends sats out of a channel in exchange for on-chain bitcoin returned to the Institute's wallet, freeing inbound capacity in the process.

Liquidity is also the reason a node earns routing fees at all. Balance sitting idle in a channel is capital tied up; routing fees are its return. A routing node's fees are, in effect, the rent on its liquidity — which is why fee policies change as balances drift.

> **Sizing it for the Institute**

> The blueprint’s financial model leans heavily on philanthropy, and most of that money will arrive as a handful of large grants that should go on-chain to cold storage, not through channels. The Lightning-sized flows are the small and recurring ones: individual donations, dues, CPD fees, contractor payments. If those run at, say, USD 50,000 a month, receiving reliably needs inbound capacity of a comfortable multiple of the largest expected payment on each inbound channel and a total of perhaps one to two weeks’ flow — tens of thousands of dollars, not millions. The number is illustrative, but its order of magnitude is the point: Lightning liquidity for the Institute is a working-capital question, measured in weeks of flow, not a treasury question.

---

## Part 7 — What can go wrong

An honest account of Lightning has to include the ways it fails. Most are operational rather than cryptographic, and each has a direct consequence for the design in Part 9.

**On-chain fees at the wrong moment.** Every safety mechanism in Parts 2 to 4 — the penalty transaction, claiming an HTLC on-chain — ultimately depends on getting a transaction confirmed before a timelock expires. If on-chain fees spike just when a node needs to force-close, the node has to outbid everyone else for block space, and a small channel can become uneconomic to close at all. Anchor outputs help; they do not abolish the problem.

**Replacement cycling.** Disclosed by Antoine Riard in October 2023 and assigned CVEs against all four major implementations, this attack uses Bitcoin's fee-replacement rules to keep a routing node's timeout transaction out of the mempool until the upstream HTLC expires, so the attacker can claim both sides. Mitigations — aggressive rebroadcasting, mempool monitoring for the preimage, longer timelock deltas — raise its cost but do not close it, and no consensus-level fix has been deployed. No verified theft has been reported. It is the strongest argument against running very short deltas on large channels.

**Channel jamming.** A channel can carry at most 483 pending HTLCs in each direction. An attacker can send payments to itself that it never settles, occupying those slots or the channel's liquidity until the timelocks expire, at almost no cost. This disrupts rather than steals. Mitigations — reputation systems and up-front fees — are being designed and tested rather than finished.

**Operator error.** By some way the commonest cause of real losses: restoring a node from an out-of-date backup (which broadcasts a revoked state and triggers your own peer's penalty transaction against you), letting a node go offline for longer than `to_self_delay`, or mismanaging force-closes. The cure is boring: well-tested infrastructure, static channel backups, watchtowers, monitoring and trained staff.

**The quantum horizon.** In The Spinning Coin and the Stolen Key I explained why Shor's algorithm breaks secp256k1, the elliptic curve Bitcoin uses for its signatures. Lightning is more exposed than ordinary on-chain storage, because the public keys behind every public channel's funding output are broadcast to the whole network in channel announcements, and Taproot outputs put the key on-chain at once. Long-lived funds in public channels are therefore among the first targets a cryptographically relevant quantum computer would have. Proposals for post-quantum Bitcoin outputs are being discussed; none is activated. For Malipo Rail the conclusion is the same as for the blockchain under MedLattice: keep the exposure short-lived and small, and keep long-term reserves out of channels.

## Part 8 — From satoshis to shillings: M-Pesa, by the same method

Everything so far moves bitcoin. The contractor in Mombasa wants shillings on her phone. That means crossing from a system built on hashes and timelocks into one built on an entirely different model — and the crossing is where most of the engineering, and most of the risk, actually lives.

### What M-Pesa is, mechanically

M-Pesa is not a bank and not a blockchain. It is an **electronic-money ledger** run by Safaricom, in which every shilling of customer balance is backed one-for-one by money Safaricom holds in trust accounts at regulated Kenyan banks, under the Central Bank of Kenya's supervision. A payment is an entry in Safaricom's ledger: one account down, another up, final the moment the ledger commits it.

Its scale is the reason this section exists. In Safaricom's 2026 financial year, to 31 March, M-Pesa processed **46.4 billion transactions worth KES 41.68 trillion**, had **40.99 million** one-month active customers in Kenya and **333,011** agents, and served about 3.1 million merchant businesses. Its platform handles 6,000 transactions a second, with headroom to scale past 12,000. Lightning, for comparison, carried a little over US$1 billion in November 2025, by one estimate, across the whole world. In Kenya, M-Pesa is not a competitor to be displaced. It is the ground the rail has to stand on.

The rules that matter for a payment rail:

- **Limits.** KES 250,000 per transaction, KES 500,000 per day and a maximum balance of KES 500,000 for an individual account, set under CBK oversight. A large donation cannot simply be paid out to a phone.
- **Merchant pricing.** Since August 2026, "Buy Goods" payments to a till are free to the merchant up to KES 500, and cost 0.55% above that, capped at KES 200. Most patient bills fall in this range.
- **Finality and reversal.** An M-Pesa payment is final in Safaricom's ledger immediately, but Safaricom operates a reversal process for payments sent in error, and the business API exposes a reversal call. From a bridge's point of view, an incoming M-Pesa payment is fast but not as irreversible as a settled Lightning payment.

### The Daraja API

Businesses connect to M-Pesa through **Daraja**, Safaricom's REST API. The calls a payment bridge needs:

- **STK Push** ("Lipa na M-Pesa Online"): the business triggers a payment prompt on the customer's phone; the customer enters their PIN; the result arrives asynchronously at a callback URL.
- **C2B** (customer to business): payments the customer initiates to a paybill or till. The business registers a *validation* URL, called before the payment completes and able

to reject it, and a *confirmation* URL, called after it completes with Safaricom's transaction ID.

- **B2C** (business to customer): payouts from a business shortcode to a phone number. The request carries a unique `OriginatorConversationID`; the result arrives later at a `ResultURL`, or a timeout notice at a `QueueTimeOutURL`.
- **B2B**: transfers between business shortcodes — the call a clinic would use to pay the Institute's paybill.
- **Transaction Status** and **Reversal**: for checking what actually happened and undoing an error.

The feature that matters for everything that follows is that **every money-moving call is asynchronous**. You ask; some seconds later Safaricom tells you the result, by calling you back. Lightning is asynchronous too, but in a very particular way: an HTLC can wait, provably and safely, for an external event, up to its timelock. That is the hook.

### The bridge problem

Take the contractor's payout. The Institute pays a bridge provider over Lightning; the bridge pays the contractor's M-Pesa number with a B2C call. The naive design does these in sequence — receive the Lightning payment, then send the shillings — and has a gap: if the B2C payout fails, the bridge has already taken the bitcoin and must be trusted to refund it. The reverse design — pay out shillings first, then collect — puts the risk on the bridge instead. Either way, someone is trusting someone, which is exactly what the Lightning half of the system was built to avoid.

### The hold invoice: making the bridge almost atomic

A **hold invoice** is an ordinary Lightning invoice with one difference: the recipient does not settle the incoming HTLC the moment it arrives. It *holds* it — the money is locked in the channel, provably on its way, but not yet claimed — until the recipient decides to settle it by revealing the preimage, or to cancel it, returning the money. The holding is bounded by the HTLC's timelock, which for the final hop is at least tens of blocks — hours, not seconds.

Wire that to Daraja and the bridge becomes:

1. The bridge generates a preimage and issues a **hold invoice** for the Lightning amount, quoting KES 25,800 to the contractor's number at a stated exchange rate.
2. The Institute pays it. The HTLC arrives at the bridge and is **held**. The bridge now knows, cryptographically, that 250,000 sats are committed to it — but it cannot yet take them, and the Institute's node can see that they have not been taken.
3. The bridge sends a **B2C** request to Daraja with a fresh `OriginatorConversationID`.
4. **If the `ResultURL` callback reports success**, with Safaricom's transaction ID, the bridge **settles** the HTLC by revealing the preimage. The Institute's wallet receives the preimage — proof of payment — and the bridge receives the sats.

5. **If the callback reports failure, or the request times out,** the bridge first queries **Transaction Status** to make sure the payout really did not happen, then **cancels** the HTLC. The sats return to the Institute along the route.

There is no longer a window in which the bridge holds the Institute's money and the contractor has nothing. The Institute's money only becomes the bridge's at the moment the bridge has evidence that the shillings have landed.

> **"Almost" atomic, not atomic**
>
> This is not a true atomic swap, and it should not be described as one. The M-Pesa leg is not cryptographically bound to the Lightning leg; it is bound by the bridge's software reading Safaricom's callback. A bridge that lies — settles the HTLC without sending the shillings — can still steal one payment. What changes is the *shape* of the trust: the bridge can no longer accidentally keep money for a payout that failed, every completed payment produces both a Lightning preimage and an M-Pesa transaction ID that can be matched, and the remaining risk is plain dishonesty by a licensed, identifiable business, which is the kind of risk contracts and regulators are good at. The opposite direction — shillings in, sats out, as when a donor inside Kenya pays by M-Pesa — is harder, because the M-Pesa payment arrives first and can in principle be reversed; there the bridge carries the risk and prices it in.

## The incumbents already doing this

None of this is hypothetical. In May 2026 Tando made every Kenyan M-Pesa number usable as a Lightning address of the form `0712345678@bitcoin.co.ke`: sats sent to an unclaimed address are converted and paid to the recipient's M-Pesa for a 1% fee, while a recipient who verifies the number once (for KES 210) receives sats directly, for transfers between KES 15 and 10,000. Through Fedi's mini-app catalogue, Tando also lets Lightning users pay M-Pesa phone numbers, "Buy Goods" tills and paybills, with the merchant seeing only an ordinary M-Pesa payment. Bitnob reports payout corridors across more than twenty countries and strong growth; Machankura brings Lightning to feature phones over USSD. The bridge in this section is, in other words, a description of how such services can be built safely, not a proposal to build one from scratch.

## The comparison I promised in September

My draft's research agenda said: "Before committing engineering time, we need real numbers on where Lightning genuinely beats the incumbent … Do not assume the answer." Here are the numbers.

| Payment | M-Pesa | Lightning (+ bridge where needed) |
|---|---|---|
| Patient pays a KES 400 bill at a till | Free to both sides | Pointless: the patient would need bitcoin first |
| Patient pays a KES 2,400 bill at a till | KES 13.20 to the merchant (0.55%) | ~KES 24 via a 1% conversion bridge, plus the patient needing sats |

| Payment | M-Pesa | Lightning (+ bridge where needed) |
|---|---|---|
| Clinic pays KES 25,000 dues to the Institute | KES 52 flat from a till to a paybill (August 2026 tariff) | Sub-shilling routing fee *if* both hold sats; otherwise conversion costs dominate |
| Donor in Toronto sends US$200 | Not available directly; needs a remittance service or international card | ~176 sats (KES 18) in routing fees, under a second, final; plus conversion if paid out in KES |
| Institute pays a contractor KES 25,800 on M-Pesa | B2C tariff agreed with Safaricom for the Institute's shortcode | Routing fee + ~1% bridge fee (~KES 258), with proof of payment |

The verdict is not close, and it is not what I implied in September. **For domestic payments between people who hold shillings, M-Pesa wins.** It is cheaper, universal, already trusted, and adding Lightning in the middle only adds a conversion cost and a new failure mode. Lightning wins where M-Pesa does not reach: **money arriving from outside Kenya**, payments between parties who already hold bitcoin, and situations where a cryptographic proof of payment is worth more than a few shillings of fee.

## Part 9 — The Malipo Rail, re-specified

Here is the design as it now stands, with each flow mapped onto the mechanics above and each change from the September draft marked.

| # | Flow | Rail | Lightning mechanics used | Change since September |
|---|---|---|---|---|
| 1 | **Patient bills** at member hospitals | M-Pesa, directly to the hospital's own till or paybill | None | **Changed.** Was a Lightning invoice priced in KES |
| 2 | **Small and recurring donations**, especially from abroad | Lightning, to the Institute's own node via a BOLT12 offer or ordinary invoice | Offers and blinded paths; multi-path payments; inbound liquidity bought from an LSP | Refined. Does not rely on BOLT12 recurrence |
| 3 | **Large grants** | On-chain bitcoin, directly to the cold multisignature treasury | None | Unchanged |
| 4 | **Milestone disbursement** from the treasury | On-chain from the multisig, authorised by `DonorFundAllocator` events; converted to KES through a licensed partner | None | Unchanged |
| 5 | **Hospital dues and CPD fees** | M-Pesa paybill by default; Lightning for members who already hold bitcoin | Ordinary invoices | **Changed.** Was Lightning by default |
| 6 | **Contractor payouts** | Lightning; to M-Pesa through a licensed bridge using a hold invoice | Hold invoice wired to a Daraja B2C callback | New detail |
| 7 | **Settlement between institutions** (member hospitals, partner institutes abroad) | Direct Lightning channels between the institutions' nodes | Channels, HTLCs, splicing to resize | New |
| — | **Every completed payment** | Attested on MedLattice's chain | Preimage-based commitment (below) | Specified |

### Three corrections to my own draft

**1. Patients should not touch Lightning at all.** The draft had the hospital's point of sale generating Lightning invoices priced in shillings, with bitcoin "invisible to the patient". Part 8's arithmetic shows that even invisible bitcoin costs the patient more than paying the till directly. Patients pay in M-Pesa to the hospital's own till or paybill. Lightning appears, if at all, only behind the hospital's finance office.

**2. BOLT12 does not do recurring payments.** The draft said dues and fees were "well suited to Lightning's BOLT12 recurring-offer pattern". The BOLT12 standard merged in 2024 gives a reusable payment code; a *recurrence* extension has been proposed, not standardised. A recurring donation today means the donor's wallet paying the same offer each month, which some wallets automate and many do not. The design should not depend on it.

**3. The dollar stablecoin leg is less attractive than it was.** The draft suggested Tap-

> root Assets USDT as a way to price in a stable unit. The technology has matured — USDT is live on Lightning via Taproot Assets, routed as ordinary bitcoin HTLCs between asset-aware edge nodes that quote a rate — but the VASP Regulations, 2026, give the Central Bank of Kenya power to control the availability of foreign dollar-linked stablecoins. A Kenyan institution building on a foreign dollar stablecoin is now building on something its central bank has explicit power to restrict. The stable leg should be shillings via a licensed Kenyan partner.

### The node, the LSP and the treasury

The Institute runs **its own Lightning node** — not a custodial account — because a node that holds its own keys is the only way to make the proof-of-payment and attestation properties below true. It does not try to be a major routing node. It opens a small number of large, well-maintained channels to two or three reliable counterparties, including at least one **LSP** from which it buys inbound liquidity, and at least one licensed Kenyan bridge for shilling payouts. Its software is one of the mature implementations, chosen in part for BOLT12 support (Core Lightning, LDK or Eclair natively, LND only via an add-on) and splicing. It is monitored around the clock and registered with at least two independent watchtowers.

Channel balances are **working capital**, sized in weeks of flow, and swept regularly to the **treasury**: cold, multisignature, on-chain storage, in script-hash outputs that are not reused, with keys held by named signers in more than one jurisdiction. Nothing that the Institute holds for the long term ever sits in a channel — for the operational reasons of Part 7 and for the quantum reason. Large grants arrive on-chain, directly to the treasury.

### What the attestation registry records

The September draft proposed a `PaymentAttestationRegistry` on MedLattice's permissioned Besu chain, "recording a cryptographic commitment" for every payment, and left the contents vague. Working through Lightning makes them precise, because Lightning already produces exactly the right object: **the preimage is the receipt.** Only the payee could have revealed it, and only by accepting the payment.

For each completed payment, the registry stores a single 32-byte commitment:

```
C = SHA3-256( payment_hash ‖ preimage ‖ amount_msat ‖ settled_at ‖ payer_compartment_ref
‖ payee_compartment_ref ‖ mpesa_txid? ‖ salt )
```

Three properties follow. **Anyone holding the inputs can verify the payment** — the hospital, the donor's auditor, the Institute's finance office — by recomputing the hash and finding it on the chain. **Nobody without them learns anything**: the salt stops anyone hashing guesses ("did patient X's compartment pay amount Y?") against the registry. And **the two worlds stay separate**: the registry holds compartment references,

never wallet addresses or identities, and the Lightning network never sees a compartment reference. For a bridged payment, Safaricom’s transaction ID goes into the same commitment, binding the shilling leg to the bitcoin leg in one record.

### What does not change

The architectural separation in the September note is, if anything, stronger for having worked through the mechanics. No bitcoin, no wallet address, no private key and no patient-identifying information ever touches MedLattice’s chain. The payment system sees amounts, hashes and channels; the clinical record sees compartments and consents. They meet only in a salted commitment that neither can open alone. A compromise of the Institute’s Lightning node cannot read a single patient record; a compromise of MedLattice cannot move a single satoshi.

## Part 10 — The Kenyan regulatory picture, two days before the deadline

The September note described the Virtual Asset Service Providers Act, 2025, and its Regulations. Since then:

- **The Regulations are in force** (Legal Notice 134 of 2026, in effect from late July), and the transitional period for businesses already operating ends on **4 November 2026** — the day after tomorrow. As of the end of October, published trackers report that **no exchange or wallet provider yet appears on a regulator's list as licensed** under the Act.
- **The Central Bank of Kenya has power to control the availability of foreign dollar-linked stablecoins**, which is why the stable leg in Part 9 is now shillings rather than USDT.
- **The Act regulates businesses providing virtual-asset services, not individuals holding them.** That helps patients and donors. It does not resolve the Institute's own position.

> **D-01 is still open**
>
> Nothing I have found addresses whether an institution that runs its own non-custodial Lightning node — holding only its own funds, never a customer's — and keeps donated bitcoin in its own multisignature custody is providing a "virtual asset service". Working through the mechanics sharpens the question rather than answering it. In Part 9's design the Institute never holds anyone else's money: patients pay in shillings, the bridge is a licensed third party, and channel balances belong to the Institute. That is the strongest version of the argument that the Institute is a *user* of the system rather than a provider of it. It is still an argument, not an opinion of counsel. Nothing should move until a Kenyan VASP lawyer has given one.

## Part 11 — The decisions log, updated

The September draft listed twelve open decisions. Here is where each stands after this work, plus two new ones.

| No. | Decision | September | Now |
|---|---|---|---|
| D-01 | Licensing of a self-custodial treasury and node | Open; needs counsel | **Open.** Design narrowed so the Institute never holds third-party funds; opinion still required |
| D-02 | Lightning infrastructure model | Self-hosted, LSP or partner | **Own non-custodial node + LSP for inbound liquidity + licensed bridge for KES** |
| D-03 | Stablecoin leg | Taproot Assets USDT or licensed partner | **Licensed KES partner.** USDT deprioritised given CBK's powers over foreign stablecoins |
| D-04 | Treasury multisig | Open | Open; now explicitly cold, on-chain, script-hash outputs, never in channels |
| D-05 | `ClaimsAdjudicator` and pedagogical compliance | Open; risky | Open; unchanged. Still the riskiest idea in the design |
| D-06 | Non-transferable internal credit | Open; needs counsel | Open; unchanged |
| D-07 | KES/BTC float policy | Open | **Sweep channel balances above a working-capital ceiling to the treasury; convert operational needs to KES via the bridge** |
| D-08 | Retention conflict | Open | Open; attestation commitments carry no personal data, which narrows but does not remove it |
| D-09 | Patient wallet custody | Custodial-lite via partner | **No patient wallet.** Patients pay in M-Pesa |
| D-10 | Ethereum/L2 stablecoin channel for donors | Deferred | Deferred; harder to justify under the stablecoin powers |
| D-11 | Naming | Placeholder | Placeholder |
| D-12 | Who operates it | Open | Open; Part 7 makes clear it needs a trained, on-call operator from day one |
| **D-13** | **Bridge risk allocation** | — | **New.** Who bears loss if a bridge settles a hold invoice without paying out; contractual terms with the licensed bridge |
| **D-14** | **Quantum exposure of channels** | — | **New.** Maximum channel size and lifetime; migration plan when post-quantum outputs exist |

## What this does not do

This piece explains how Lightning works and how a payment rail for the Institute could use it. It does not establish that the Institute should build one. The comparison in Part 8 makes the case for Lightning narrower than my September note did, and the honest summary is that the strongest uses — donations from abroad, contractor payouts, inter-institution settlement — are real but modest in volume compared with what M-Pesa already does well.

The worked examples simplify. Real commitment transactions use per-commitment keys derived by elliptic-curve arithmetic, not a single "revocation secret"; real HTLC scripts have separate second-stage transactions for timeouts and successes; real routing adds random "shadow" blocks to timelocks to protect privacy; and real fee computations include rounding rules I have skipped. None of these change the logic; all of them matter to anyone implementing it.

The probability figures in Part 6 come from the simplest model — uniform, independent balances — and real networks deviate from it in both directions. The liquidity sizing is an order-of-magnitude illustration, not a budget. The bitcoin price used throughout is illustrative.

The descriptions of Tando, Bitnob and other operators are drawn from their own publications and press coverage; I have not audited them, and naming them is not a recommendation. The regulatory summary describes published law and secondary commentary as of the end of October 2026 and is not legal advice.

---

## Where this leaves the argument

In September I said Bitcoin and Lightning earned their place in the Institute's design "by elimination as much as by their own merits". Having worked through the mechanics, I would put it differently.

Lightning is a remarkable piece of engineering. From three primitives — a hash, a clock and a signature — it builds a way for strangers to pay each other across a network none of them controls, in under a second, without trusting anyone on the path. The penalty mechanism, the HTLC chain and the onion are each simple enough to work on paper, and together they do something no payment system before them could.

But a payment system's value depends on where it sits, and in Kenya it sits beside M-Pesa — forty million people, roughly a third of a trillion dollars a year, and a merchant fee that is now zero below 500 shillings. The right design does not try to replace any of that. It lets patients and clinics keep using what already works, and uses Lightning for the things M-Pesa cannot do: bringing money in from outside Kenya, paying people with cryptographic proof, and moving value between institutions without either trusting the other. And it keeps the Institute's reserves where Bitcoin is strongest — in cold storage

on the base chain, out of reach of channel failures, quantum horizons and anyone's operational mistakes.

That is a smaller claim than the one I made in September. I think it is also a truer one, and it is what I am now building.

---

## A short glossary

**Anchor output.** A tiny output on a commitment transaction that lets either party add a fee at broadcast time.

**BOLT.** "Basis of Lightning Technology" — the numbered documents that specify the Lightning protocol.

**BOLT12 offer.** A reusable, static Lightning payment code that generates a fresh invoice for each payment, using blinded paths for receiver privacy.

**Channel.** A 2-of-2 multisignature output plus a succession of signed commitment transactions dividing it between two parties.

**Commitment transaction.** A signed but unbroadcast transaction representing the current balance of a channel.

**Daraja.** Safaricom's REST API for M-Pesa: STK Push, C2B, B2C, B2B, Transaction Status, Reversal.

**Force close.** Unilateral broadcast of the latest commitment; the broadcaster's own funds are delayed by `to_self_delay`.

**Hold invoice.** An invoice whose recipient holds the incoming HTLC rather than settling it immediately, until an external condition is met.

**HTLC.** Hash time-locked contract: pays the next party if they reveal a hash preimage before a block height, otherwise refunds.

**Inbound liquidity.** Balance on the far side of your channels; what lets you receive.

**LSP.** Lightning Service Provider: a node that sells channels and liquidity to others.

**Preimage.** The secret whose hash locks a payment; revealing it completes the payment and serves as proof of payment.

**PTLC.** Point time-locked contract: a proposed replacement for HTLCs that uses a different lock at each hop, improving privacy.

**Revocation.** Handing the other party the secret for an old state, so that broadcasting it can be punished.

**Splicing.** Changing a channel's size without closing it.

**Timelock delta.** The safety margin in blocks a routing node requires between its incoming and outgoing HTLCs.

**Watchtower.** A third-party service that watches the blockchain for revoked states on a node’s behalf and broadcasts the penalty.

---

## Further reading and references

### The protocol


- Poon J, Dryja T. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments.* 2016. The original paper.
- The BOLT specifications — especially BOLT 2 (channel management), 3 (transactions), 4 (onion routing), 7 (gossip), 11 (invoices) and 12 (offers).
- Pickhardt R, Richter S. Optimally Reliable & Cheap Payment Flows on the Lightning Network. arXiv 2107.05322, 2021.
- Spark — CLTV delta (implementation defaults) and Replacement cycling.
- Spark — State of the Lightning Network in 2026, June 2026 (network statistics, protocol status; Spark is a competing layer 2).
- Spark — Taproot Assets: issuing tokens on Bitcoin and routing them over Lightning.


### M-Pesa and Kenya


- Paybill KE — Kenya mobile money statistics 2026 (from Safaricom's FY2026 annual report) and M-Pesa transaction limits 2026.
- TechTrends KE — M-Pesa handled KSh41.68 trillion in FY2026, 28 October 2026.
- The Standard — Safaricom halves M-Pesa merchant fees, 2 August 2026.
- Paybill KE — M-Pesa Daraja API complete guide 2026; McTaba — Daraja C2B, B2C and B2B.
- Blink — 40 million Kenyans now have a bitcoin Lightning address, 22 May 2026; Fedi — Spend bitcoin in Kenya and have it settle over M-Pesa with Tando, 23 March 2026.
- Paybill KE — Kenya VASP licence status 2026, 25 October 2026; AU Startups — CBK powers over foreign stablecoins, 28 August 2026; Spencer West — Kenya's VASP Regulations, 2026.


### Earlier essays in this series


- What I'm Building Next: A Bitcoin Rail for the Clinical AI Institute — the September working note this piece replaces
- Not Shown Is Not Locked: The Patient Record I Am Building
- Twenty Generals, One Ledger: Why MedLattice Runs on Hyperledger Besu and QBFT
- The Spinning Coin and the Stolen Key: What Shor's Algorithm Actually Threatens
- Learning With Errors, by Hand: The Mathematics MedLattice Is Named After
- Another Arrow in the Quiver: The Clinical AI Institute Kenya Should Build